Load Shedding in Classifying Multi-Source Streaming Data: A Bayes Risk Approach

Yijian Bai
UCLA
bai@cs.ucla.edu

Haixun Wang
IBM T. J. Watson
haixun@us.ibm.com

Carlo Zaniolo
UCLA
zaniolo@cs.ucla.edu

Abstract

In many applications, we monitor data obtained from multiple streaming sources for collective decision making. The task presents several challenges. First, data in sensor networks, satellite transmissions, and many other fields are often of large volume, fast speed, and highly bursty nature. Second, because data are collected from multiple sources, it is impossible to offload classification decisions to individual data sources. Hence, the central classifier responsible for decision making is constantly under overloaded situations. In this paper, we study intelligent load shedding for classifying multi-source data. We aim at maximizing classification quality under resource (CPU and bandwidth) constraints. We use a Markov model to predict the distribution of feature values over time. Then, leveraging Bayesian decision theory, we use Bayes risk analysis to model the variances among different data sources in their contributions to classification quality. We adopt an Expected Observational Risk criterion to quantify the loss of classification quality due to load shedding, and propose a Best Feature First (BFF) algorithm that greedily minimizes such a risk. We also introduce an approximate BFF algorithm that reduces computation complexity. The effectiveness of the approach proposed is confirmed by several experiments on both synthetic and real-life data.

1 Introduction

Mining high-speed, large volume data streams introduces new challenges for resource management [8, 11]. In many applications, data from multiple sources (e.g., data collected by different types of sensors) arrive continuously at a central processing site, which analyzes the data for knowledge-based decision making. Typically, the central site handles a multitude of such tasks at the same time, which makes resource management a major issue for many applications. In particular, under overloaded situations, policies of *load shedding* must be developed for incoming data streams so that the quality of decision making is least affected. In this paper, we develop principles of load shedding for multi-task, multi-source stream classification applications.

Multi-task, Multi-source Stream Classification. Consider a central sever that handles n independent classification tasks, where each task processes a multiple number of input streams. Figure 1 shows a typical configuration of the system, where, for presentation simplicity, we assume all tasks have k input streams.

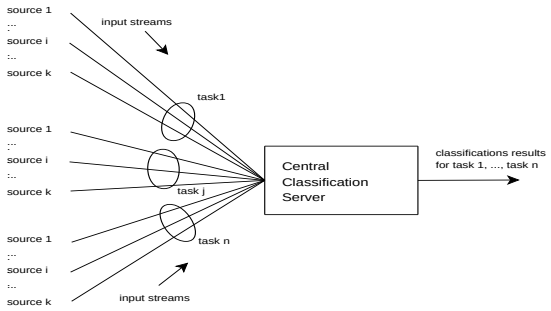


Figure 1: Multi-task, multi-source stream classification

Our problem is the following. *Suppose at a given moment, the central classifier, which monitors $n \times k$ streams*

from n tasks, only has capacity to process m out of the $n \times k$ input streams. Then, which of the input streams should be inspected so that the classification quality is least affected?

We use the following two examples to illustrate situations that give rise to the problem.

1. A security application monitors many locations with security cameras. At each location, multiple cameras are set up at different viewing angles, since the speed and direction of a moving object cannot be determined precisely if only one viewing angle is used. As a result, each location generates multiple image or video streams and sends them to the central server for classification. In this case, data from different cameras are of the same type but they have different semantics (different viewing angles).
2. In environment monitoring, a central classifier makes decisions based on a set of factors, such as temperature, humidity, wind-speed, etc., each obtained by sensors distributed in a wireless network over a wide geographical region. In this case, multiple data sources for one task contains different types of information.

An inherent challenge to the problem is that the central task of decision making cannot be easily offloaded to each data source, as classification depends on information from all of the k sources. On the other hand, in most situations, it is safe to assume that at any given time, there exist only a small number of events of potential interest, which means, even if $m \ll n \times k$, it is still possible to monitor all the tasks and catch all events of interest if we know how to intelligently shed loads.

Upon receiving the data from multiple sources, the central server fuses them into a single form that is processable by the classifier. For ease of discussion, we assume the combined data is a vector, and each source provides data as one *feature* of the vector[1].

The goal of intelligent load shedding is to reduce the cost of the stream classification process while maintaining the quality of classification. The following factors may have significant implications on the overall cost.

- Cost of data preprocessing. Raw data from the sources may have to be preprocessed before classification algorithms can be applied. For example, for video streams, extracting frames from a video and extracting features from key frames can be a very costly process.
- Cost of data transmission. Delivering large amount of data from remote sources to the centralized server may incur considerable cost.
- Cost of data collection. Data may be costly to obtain to begin with. This may limit the sampling rate of a sensor, or its on-line time due to energy conservation concerns.

As a concrete example, the central server in the above security application may have to perform a two-step procedure: a) the server *observe* the video stream, i.e. receive the stream from the network, parse the video frame images to determine the composition of objects and the location of objects in the image. This step has a very high computation cost in terms of data transmission and video parsing. b) the server runs a classification algorithm on the interpreted image to determine potential security risk.

Cost is reduced if high quality decisions can be made with less amount of data. For example, if we can predict the object locations in the video based previous data, then we can avoid parsing incoming video frames. Granted, there might be extra overhead associated with making *intelligent* decisions on load-shedding. However if the any of the above costs are the dominant factors in the classification process, it becomes worthwhile to pay a reasonable cost for *intelligence* to avoid the full costs of data acquiring and observations. For example, the cost of simultaneously receiving and interpreting streaming videos from many monitoring video-cameras can become prohibitively expensive. Therefore avoiding such data observations by prediction has the potential to reduce the central CPU and network load tremendously. In this paper, we study this problem of *intelligent* load-shedding in detail.

State-of-the-Art Approaches. Although stream classification has been a focus of recent study, none of the existing solutions fully address the challenges associated with our problem.

- *Randomly shedding load*
  While dropping data indiscriminately and randomly

[1] Each input stream may in fact contain one or more *features* for the classification algorithm.

from incoming data streams is an obvious choice [9, 3, 2], such methods lead to degradation of classification quality. In many cases, not all incoming data contribute equally to the overall quality of classification. In overloaded situations, it is much more desirable if we can somehow choose to drop data that contributes the least to the quality.

- *Relying on static QoS metric functions*
  Static QoS specifications [9] assumes that the user has *a priori* knowledge about how data contributes to the quality. In other words, a data source can apply a QoS metric f on a data item $\vec{x} = (x_1, \cdots, x_n)$, and the value of $f(\vec{x})$ indicates to what extent dropping $\vec{x}$ negatively impacts the quality. Unfortunately, stream applications operate in a dynamically changing environment, which means f is unlikely to stay static. For example, data classification applications must handle concept drifts, where the decision model has to constantly evolve to adapt to the changing data distribution [18]. The multi-source setting introduces more restrictions in using such static QoS – even if we have a metric for the collective $\vec{x}$, we may not have metrics for each component x_i of $\vec{x}$, which means sources still cannot drop the data.
- *Solving the special case of $k = 1$*
  LoadStar [4] focused on a special case of our problem. It assumes each classification task has only one data source ($k = 1$). At any time, it decides which task to work on. Thus, the load shedding decisions are made on a task-by-task basis and it does not take into consideration the fact that different features of the data may contribute differently to the overall quality. In fact, for $k = 1$, we can safely offload classification tasks from the centralized server to each data source, which already has complete information to make load shedding decisions. However, for multi-source classification tasks, load shedding cannot be offloaded to the source, as only the central server has complete information about each task.

Observations. In this paper, we introduce a quality-aware load shedding mechanism based on the following observations.

1. Streaming data often exhibit strong temporal-locality (e.g. videos showing objects' movement over time). This property enables us to build a model to predict the content of the next incoming data. It then enables us to predict the classification result in the near future. If this can be done with high confidence, then we can skip observing data sources that are computationally expensive to process (e.g. video streams).

2. It is important to recognize that, at a given time, multiple sources (features) of a task (e.g. multiple cameras) may have different degree of impact on the classification result. For example, an approaching object may be caught by a camera at one angle much earlier before being caught by another camera. Thus, it is advantageous to perform feature-based load shedding, which is the focus of this paper, instead of task-based load shedding (as LoadStar [4] performs). In other words, for a given task, we should only observe those features that contribute the most to classification accuracy. Then, among a number of different classification tasks, we pick to observe a combination of features across different tasks to maximize the total classification quality.

3. In the Bayesian decision theory, *Bayes Risk* is used to measure the quality of classification, and prevent the Bayesian classifier from certain types of errors that are costly for a particular application [5] (e.g. getting false alarms in a security application is usually more acceptable than missing alarms). In this paper, we argue that it is only natural that the same criteria should be considered on the data acquisition level as well, i.e. if the classifier tries to prevent a certain error, then the load-shedder for the classifier must try to prevent the same error when making load shedding decisions, instead of using another unrelated load-shedding criteria.

   To achieve this, a simplistic method would use Bayes Risk as the guideline for load shedding, i.e., greedily observing those features that lead to the largest reduction of expected Bayes Risk. In this paper, we show that the optimal guideline is embodied by part of the expected Bayes Risk (which we term expected Observational Risk) caused solely by the lack of data observation. With this new optimization goal, we achieve good classification quality in the presence of aggressive feature-based load shedding. The

adopted Expected Observational Risk precisely captures the portion of total Bayes Risk that are caused by the lack of data value observation, therefore is more effective as an optimization goal for load shedding than the full Bayes Risk.

Contributions. To the best of our knowledge, this is the first report in the literature that studies load shedding for the general multi-source stream classification problem. Our paper makes the following contributions. (a) We propose *feature-based* load shedding for stream classification, using Observational Risk as the guideline for load shedding. (b) We give a complete analysis of the Bayes risk and a novel algorithm BFF that greedily minimizes the expected Observational Risk on a feature-by-feature basis. (c) We present an alternative algorithm that approximates the BFF algorithm, which trades some decision quality for reduced cost. (d) We present experiments on both synthetic data and real-life data to show the advantage of our algorithm over random load shedding and *task-based* load shedding methods.

Paper Organization. The rest of the paper is organized as follows. Section 2 analyzes the general problem of multi-source stream classification, and motivates the approach of using Markov model to predict data values for load shedding. Section 3 analyzes Bayes risk and Observational Risk. Section 4 presents the feature-based load shedding algorithm Best Feature First (BFF), and its lower-cost variant HVWT. Section 5 shows BFF's gain in performance over random and task-based algorithms.

2 Problem Analysis and the Markov Model

2.1 Problem Analysis

Assume a classification task monitors two data sources X_1 and X_2 for threats. Each of the sources sends a single feature stream. Thus, at any time t, the state of a task can be modeled as a point in a two-dimensional feature space. In Figure 2, we show states for three such tasks at time t, which we denote as $A(t)$, $B(t)$, and $C(t)$. Furthermore, we assume the feature space is divided into two areas such that points in the shaded area represent threats, and points in unshaded area represent non-threats.

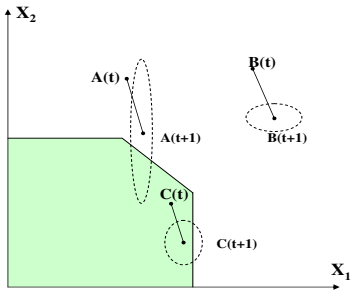


Figure 2: Task Movement in the Feature Space

Let p be the probability distribution of a point's position at time $t+1$ given its position at time t. The example in Figure 2 illustrates p as a normal distribution and it also assumes that the two features X_1 and X_2 are independent. It follows that the position of a point at time $t+1$ is within an elliptical boundary with high probability. Knowing the distribution p enables us to form some load shedding strategies, which can be used to guide our data observation (e.g. feature extraction, video analysis) at time $t+1$.

- First, different tasks should be given different priorities when we make data observation. For example, according to p, the next position of B is far away from the decision boundary, so without making data observation at time $t+1$, we can already classify B with high confidence—no matter where B moves to, its classification result will stay the same with high confidence, thus we can safely predict its class label without any observation. This is not true for A and C, for which data observations are necessary for better classification accuracy.
- Second, different features (streams) should be given different priorities when we make data observation. In Figure 3(a), we zoom in on task A, where distribution p at time $t+1$ is represented by an elliptical confidence boundary. The question is, if we can only afford to make one observation, either of X_1 or of X_2, which observation should we make? Suppose we choose X_1, and the observed value x_1 happens to be the mean. Then the elliptical region degenerates into a vertical line segment in Figure 3(b), representing the conditional distribution $p(X_2|X_1 = x_1)$. However, this does not help us much – the conditional dis-

tribution runs across the decision boundary, and we are still unable to classify A with high confidence. If we instead make observation on X_2, and again we assume the observed x_2 is the mean of X_2, the resulting conditional distribution $p(X_1|X_2 = x_2)$ will not run across the decision boundary, which enables us to make a classification with a much higher confidence— i.e., with high confidence no matter what the value of X_2 happens to be, the classification will be the same.

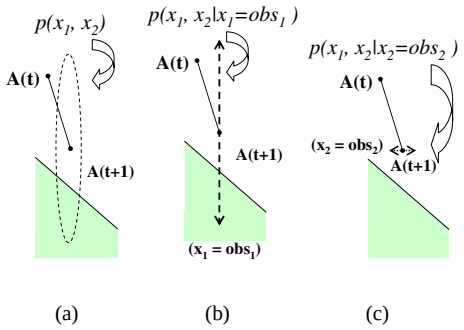


Figure 3: Joint and Conditional Distributions

In summary, from p we derive the following guideline for load shedding: for task A, it is more beneficial to observe X_2 than X_1; for C, X_1 than X_2; for B, neither observation is critical for classification. These intuitions are explained analytically by the risk analysis in Section 3.

2.2 Markov Model for Movement Prediction in the Feature Space

It is clear from our discussion in the previous section that, to make intelligent load shedding decisions at time t, we need to know p, the distribution of a point's position at time $t+1$. In other words, we should capture the temporal-locality of the data, and model the movement of a point in the feature space.

We make a simplifying assumption: a point's location in the feature space at time $t+1$ is solely dependent on its location at time t. We then build a finite discrete-time Markov chain to model a point's movement as a stochastic process.

We also assume that features are independent to each other with regard to points' movement in the feature space. This assumption allows us to build a Markov model on each feature[2]. More specifically, let X be a feature that has *M* distinct values[3]. Our goal is to learn a state transition matrix K of size $M \times M$, where entry K_{ij} is the probability that feature X will take value j at time $t+1$ given $X = i$ at time t.

We derive K through MLE (maximum likelihood estimation). The MLE of the transition matrix K is given by

$$\hat{K}_{ij} = \frac{n_{ij}}{\sum_k n_{ik}}$$

i.e., the fraction of transitions from i to j among transitions from i to k, for all possible k. In other words, we only need to use the number of observed state transitions in the history to estimate the transition matrix K. To adapt to potential concept shifts in the streaming data, i.e., to allow for the change of behavior of a point's movement in the feature space, we use a finite sliding window of recent history for maximum likelihood estimation.

3 Bayes Risk Analysis

In this section, we present a best-effort solution to the load shedding problem described in Section 1. We begin with a greedy algorithm, which is based on a naive analysis of the expected Bayes Risk over all classification tasks. Then, we argue that a portion of the expected Bayes Risk, which we call the expected *Observational Risk*, should be used as the metric for feature-based load shedding.

3.1 The Expected Bayes Risk

In Bayesian decision theory, we study the risk of misclassification by using a loss function. Let $\delta(c_i|c_j)$ denote the cost of predicting class c_i when the data is really of class c_j. Then, at a given point $\vec{x}$ in the feature space, the risk

[2]Without the independence assumption, we need a multivariate Markov model, which may requires us to estimate a very large transition matrix.

[3]We treat continuous values with the same model by discretizing them into a finite number of bins.

of our decision to label $\vec{x}$ as class c_i out of K classes is:

$$R(c_i|\vec{x}) = \sum_{j=1}^{K} \delta(c_i|c_j) P(c_j|\vec{x}) \quad (1)$$

where $P(c_j|\vec{x})$ is the posterior probability that $\vec{x}$ belongs to class c_j.

One particular loss function is the zero-one loss function, which is given by

$$\delta(c_i|c_j) = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } i \neq j \end{cases}$$

under which, the conditional risk in Eq 1 can be simplified as

$$R(c_i|\vec{x}) = 1 - P(c_i|\vec{x}) \quad (2)$$

Bayes risk is used to guide classifier training so that the learned classifier conforms with applications' error requirements. We can adjust the loss function to reflect our different tolerance to different types of errors.

We argue that the same criterion must be adopted for load-shedding. In other words, if the underlying classifier is tuned to minimize Bayes risk defined by a certain loss function, then it only makes sense that our load-shedding mechanism is optimized under the same guideline.

3.2 Expected Bayes Risk and Feature Observation

Intuition We use Figure 3 to explain what we try to achieve. We do not know the exact position of $\vec{x}$ at time $t+1$, instead, we know the distribution of its position. In Figure 3(a), the expected risk of A($t+1$) is integrated over the entire elliptical area. If, however, we choose to observe dimension X_1, the integration area is reduced to the dashed vertical line in Figure 3(b), as only dimension X_2 will have any remaining uncertainty in terms of the exact location of A($t+1$).

In other words, observations may reduce risk. But not all observations are equal. In Figure 3, if we choose to observe X_1 first, the variance of X_2 is still very high, and a large portion of the distribution is close to the decision boundary (high risk area). Whereas if we observe X_2 first, the remaining variance on X_1 is much smaller, the distribution is far away from and on the same side of the decision boundary. Therefore, observing X_2 gives a larger risk reduction.

In the rest of this section, we give a rigid analysis of the intuition we described above.

Risk Before Feature Observation Without any data observation, our knowledge about a point's next location in the feature space is given by distribution $p(\vec{x})$. The expected risk for classifying a point $\vec{x}$ as class c_i, can be represented by

$$R_{before}(c_i) = E_{\vec{x}}[R(c_i|\vec{x})] = \int_{\vec{x}} R(c_i|\vec{x}) p(\vec{x}) d\vec{x} \quad (3)$$

which is an integration over the elliptical area in Figure 3(a). Note $p(\vec{x})$ is a shorthand for $p_{t+1}(\vec{x})$, which is derived from the current distribution and the state transition matrix K of the Markov model.

$$p_{t+1}(\vec{x}) = p_t(\vec{x}) K$$

The optimal prediction c_k is the prediction that minimizes the expected risk:

$$k = \underset{i}{\operatorname{argmin}} R_{before}(c_i) = \underset{i}{\operatorname{argmin}} E_{\vec{x}}[R(c_i|\vec{x})] \quad (4)$$

Therefore the expected risk before any observation is the risk of classifying the point as class c_k, which is $R_{before}(c_k)$.

Risk After Feature Observation Suppose we choose to inspect one data stream, which supplies values for X_j. After observing $x_j = obs_j$, the total risk for labeling this partially observed data point as class c_i comes to[4]:

$$\begin{aligned} R_{after}(c_i|obs_j) &= E_{(\vec{x}|x_j=obs_j)}[R(c_i|\vec{x})] \\ &= \int_{\vec{x}|x_j=obs_j} R(c_i|\vec{x}) p(\vec{x}|obs_j) d\vec{x} \end{aligned} \quad (5)$$

Clearly, Figure 3(b) and 3(c) correspond to Eq 5 with different j's.

Risk Reduction due to Observation The benefit of making an observation of x_j is given by the reduction in the expected Bayes Risk. Suppose after observation the predicted class is c'_k, then the expected risk after observation is $R_{after}(c'_k|obs_j)$, and we have

$$R_{diff}(obs_j) = R_{before}(c_k) - R_{after}(c'_k|obs_j) \quad (6)$$

Thus, a greedy method would pick the feature that leads to the maximal reduction of risk for observation. In other words, we should choose the feature that maximizes Eq 6

[4]Sometimes we use $p(\vec{x}|obs_j)$ to stand for $p(\vec{x}|X_j = obs_j)$ for ease of presentation.

among all features from all classification tasks[5]. The best feature to observe is given by Eq 7.

$$j^* = \underset{j}{\operatorname{argmax}} R_{diff}(obs_j) \quad (7)$$

Quality of Feature Observation Eq 6 provides a guideline for feature observation in load shedding. However, in order to compute R_{diff} by Eq 6, we need to know the observed value obs_j. This contradicts our purpose: we want a metric to tell us what feature to observe.

To actually use Eq 6, we substitute obs_j by the expected value of the feature, $E[X_j]$, as our best guess for the observation. This leads to the following Quality of Observation (QoO) metric definition. The Q_{Bayes} in Eq 8 measures the quality of making an observation on feature X_j, which is conditioned upon the expected value of feature X_j.

$$Q_{Bayes}(X_j) = R_{before}(c_k) - R_{after}(c'_k|E[x_j]) \quad (8)$$

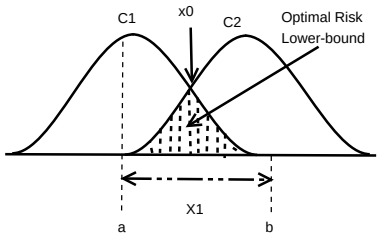
A generalized metric for making the k-th feature observation after already having observed $k-1$ features can be derived in a similar manner.

3.3 A Pitfall

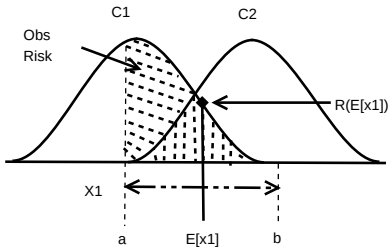
The load shedding guideline developed in the previous section is quite straightforward. However, as we demonstrate in this section, there is a pitfall in using expected Bayes risk for load shedding.

Dissecting the risk Let $p(C_1|x)$ and $p(C_2|x)$ be the posterior distributions of two classes C_1 and C_2. Without loss of generality, Figure 4(a) shows the two distributions as two Bell curves. At point x_0, we have $p(C_1|x) = p(C_2|x)$. In other words, x_0 is the classification boundary of C_1 and C_2. We further assume feature value X_1 of time $t+1$ has a uniform distribution within range $[a, b]$.

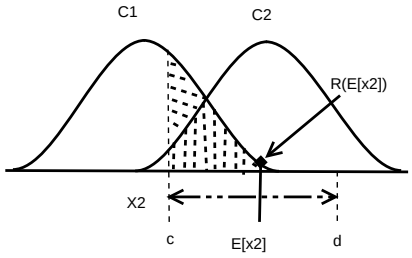
If we know $X_1 = x_1$ at time $t+1$, we can make an optimal decision, which is to predict the class that has



(a) The Optimal Risk Lower-bound of Feature X1 for time $t+1$



(b) Risk Decomposition and Expected Risk of Feature X1 for Time t



(c) Risk Decomposition and Expected Risk of Feature X2 for Time t

Figure 4: Bayes Risk Composition

[5]Note that the predicted class before any observation, c_k in equation 6, is task-dependent. I.e., the $R_{before}(c_k)$ should really be $R_{before}(c_k; task_{obs_j})$, where $task_{obs_j}$ is the task that observation obs_j belongs to. Therefore this value is shared by all observations obs_j for the same task, but different in different tasks. Same applies to equation 8.

higher posterior probability at x_1. Assuming 0/1 loss, the optimal risk at x_1 is the value of the smaller posterior probability. Therefore, given that x_1 distributes uniformly within $[a, b]$, the expected optimal risk is the average of the shaded area in Figure 4(a).

This expected optimal risk cannot be further reduced by improving the underlying classifier, or by any other means. In fact, it is the unavoidable, lowest risk, as it is dictated by the nature of the class posterior probabilities.

Then, what will be the risk if we do not know the exact value of X_1 at time $t + 1$? We still need to make a prediction, and suppose that we predict C_2. Then the total Bayes Risk is the shaded areas in Figure 4(b), and we can see the risk is not optimal at data points where C_1 should have been the optimal decision. Compared with the optimal risk, the increased portion, which we call the Observational Risk, is shown as the extra shaded areas in Figure 4(b).

The Pitfall The strategy we developed in the previous section may not be optimal in risk reduction. To see this, we can compare the two features X_1 and X_2 shown in Figure 4(b) and Figure 4(c), where we want to decide which feature to observe. As shown in the figure, X_2 has a different distribution at time $t + 1$ (uniform within $[c, d]$) from X_1, and consequently different expected value $E(X_2)$ and different Observational Risk.

By observing the value of a feature, we can eliminate the Observational Risk associated with that feature. Clearly, we should choose to observe feature X_1, because as shown in Figure 4, its area that corresponds to the Observational Risk is larger.

However, based on the strategy developed in the last section, we would opt to observe feature X_2, because it has a much lower risk value at its expected location $E(X_2)$, as shown in the figure. According to Eq 8, we should choose the feature X such that the expected risks before observing X (in the figure, the average value of the total shaded area) and after observing X (in the figure, $R(E[X])$) has the largest difference. Since $R(E[X_1])$ is much larger than $R(E[X_2])$, the risk reduction may in fact favor feature X_2.

3.4 The Expected Observational Risk

The naive risk analysis in Section 3.2 failed to deliver an optimal load shedding strategy because it tries to minimize the entire Bayes Risk when making data observation choices. It does not realize that Bayes risk consists of two parts, and only one part, the Observational Risk, can be eliminated by making observations. The other part, the Optimal Risk, is unavoidable, and data observation cannot lead to a risk lower than this lower bound.

As data observations can only reduce the Observational Risk portion of Bayes Risk, it makes sense to use Observational Risk instead of the full Bayes Risk as our optimization goal for load shedding.

In this section, we propose a new metric, Q_{Obs}, to guide data observation. The superiority of Q_{Obs} over Q_{Bayes} is due to its focus on the reducible risks, and such superiority is confirmed later in experiments in Section 5.

The Q_{Obs} Metric At each location $\vec{x}$ in the feature space, there is an optimal decision given by the underlying classifier, suppose it is c^*. Clearly, c^* is given by[6]:

$$c^* = \operatorname*{argmin}_i R(c_i|\vec{x}) \tag{9}$$

Then, we can re-write the Expected Bayes Risk for unobserved tasks in Eq 3 into the following form, where we assume the assigned class by the classifier is c_k according to Eq 4.

$$\begin{aligned} R_{before}(c_k) &= E_{\vec{x}}[R(c_k|\vec{x})] = \int_{\vec{x}} R(c_k|\vec{x})p(\vec{x})d\vec{x} \\ &= \overbrace{\int_{\vec{x}} R(c^*|\vec{x})p(\vec{x})d\vec{x}}^{\text{Optimal Risk Lower-bound}} \\ &\quad + \underbrace{\int_{\vec{x}} [R(c_k|\vec{x}) - R(c^*|\vec{x})]p(\vec{x})d\vec{x}}_{\text{Expected Observational Risk}} \end{aligned} \tag{10}$$

It is clear from Eq 10 that the expected risk for an unobserved data point consists of two parts.

- The first part, $\int_{\vec{x}} R(c^*|\vec{x})p(\vec{x})d\vec{x}$, is the expected Optimal Risk, which is the lowest possible risk that the underlying classifier can achieve.

[6] c^* is actually a function of $\vec{x}$. Here c^* is used in place of $c^*(\vec{x})$ for representation simplicity.

- The second part, $\int_{\vec{x}}[R(c_k|\vec{x}) - R(c^*|\vec{x})]p(\vec{x})d\vec{x}$, is the expected risk increase over the lower bound, which is caused by a non-optimal prediction due to classifier's lack of knowledge about the true data. This is the portion that observation of data affects the most – it is completely eliminated after the full observation of all features.

Therefore, we should first observe features that lead to the largest reduction of the second part of Bayes Risk, the Observational Risk, which only apply to un-observed (or partially observed) data.

The expectation of the Observational Risk (which we will refer to as R^{obs}) for un-observed or partially-observed data is then:

$$R^{obs}_{before}(c_k) = \int_{\vec{x}}[R(c_k|\vec{x}) - R(c^*|\vec{x})]p(\vec{x})d\vec{x} \quad (11)$$

Intuitively, if the distribution has less overlap with the decision boundary, then the Expected Observational Risk will have a lower value—this explains the guidelines derived from intuitions in Section 2.

Similarly, the risk after the first observation R_{after} in Eq 6 can also be decomposed into two parts, in much the same way as the decomposition of Eq 10 goes. Therefore the Observational Risk after observing feature x_j is given by:

$$R^{obs}_{after}(c'_k|obs_j) = \int_{(\vec{x}|x_j=obs_j)}[R(c'_k|\vec{x}) - R(c^*|\vec{x})]p(\vec{x}|obs_j)d\vec{x}$$

Now, we can replace obs_j with its expectation, and modify the Quality of Observation metric Q_{Bayes} defined in Eq 8 into Q_{Obs}, which measures the gain of Observational Risk after observing the feature X_j. (Here c_k is the predicted class before the observation, and c'_k is the predicted class after the observation.)

$$Q_{Obs}(X_j) = R^{obs}_{before}(c_k) - R^{obs}_{after}(c'_k|E[x_j]) \quad (12)$$

The above gives the guideline for picking the first feature for observation. We can use similar procedures to maximize Expected Observational Risk reductions before and after making the k_{th} feature observation for a task. Eventually, with full observation the risk is reduced to the optimal risk at the observed location $\vec{x}_{obs}$, which solely depends on the underlying classifier and the location itself, without any contribution from the data observation error.

Therefore, the generalized metric Q_{Obs} measures the quality of making the k_{th} observation x_k, which is conditioned on the feature values we have already observed so far ($obs_1, obs_2, \cdots, obs_{k-1}$), and the expected value of the feature x_k that we are about to observe.

$$\begin{aligned} Q_{Obs}(X_k) &= R^{obs}(c_k|obs_{1,\cdots,k-1}) \\ &\quad -R^{obs}(c'_k|obs_{1,\cdots,k-1}, E[x_k]) \quad (13) \\ &= \int_{\vec{x}|obs_{1,\cdots,k-1}} R'(c_i|\vec{x})p(\vec{x}|obs_{1,\cdots,k-1})d\vec{x} \\ &\quad - \int_{\substack{\vec{x}|obs_{1,\cdots,k-1}, \\ x_k=E[x_k]}} R'(c_i|\vec{x})p(\vec{x}|obs_{1,\cdots,k-1}, E[x_k])d\vec{x} \end{aligned}$$

Obviously, Eq 12 is a special case of Equation 13 where the set of already-observed features is empty.

4 The Best Feature First (BFF) Algorithm

4.1 The BFF Algorithm

The Best Feature First (BFF) algorithm (shown in Algorithm 1) is derived based on Eq 13. BFF is invoked once in every time unit, which utilizes the metric Q_{Obs} to repeatedly pick the next *best* feature to observe until the capacity for the time unit is consumed.

Intuitively, in Algorithm 1, at the beginning of each time unit we first compute the predicted distributions for each feature using Markov chains, and then compute an expected decision for each task based on the predictions. Then we repeatedly pick to observe the *best* unobserved feature over all tasks that leads to the largest reduction in Expected Observational Risk. By doing so, we greedily minimize Expected Observational Risk over all tasks.

4.2 Implementation Issues and Cost Analysis

While conceptually clear, the BFF algorithm has a few implementation and computation issues that require further elaboration.

Computing the Expected Risks: The BFF algorithm requires computing the Expected Observational Risk. For

Algorithm 1 The Best Feature First (BFF) Algorithm

inputs: A total of n classification tasks, where each task T_i has k streaming data sources(features). For the current time unit, some or all of the $N = n \times k$ streams may have new data available.
outputs: Decisions δ_i ($i \in 1, \cdots, n$) for each of the n tasks
static variables: One next feature distribution vector $p(x)$, and one Markov model K built on data in a sliding window, for each of the N streams
how to use: invoke once per load-shedding time unit

1: Compute the predicted feature distribution $p(x)$ for each feature x, based on the previous $p(x)$ value and the Markov model K.
2: Compute the predicted decision δ_i ($i \in 1, \cdots, n$) for each of the n tasks, based on the predicted feature distribution $p(x)$ (Equation 4).
3: Apply heuristics to prune the set of all features, which results in candidate feature set F_{cand}. (See discussion in Section 4.2.)
4: For all features $x_j \in F_{cand}$, compute $Q_{Obs}(x_j)$ by Eq 13
5: For all features $x_k \notin F_{cand}$, assign $Q_{Obs}(x_k) \leftarrow 0$
6: $observed\_count \leftarrow 0$
7: **while** still data and $observed\_count < Capacity$ **do**
8: Pick the unobserved stream x_j with the highest $Q_{Obs}(x_j)$ value across all features of all tasks, and observe its actual data value. Break tie randomly.
9: If the highest $Q_{Obs}(x_j)$ equals to 0, randomly pick the remaining ($Capacity - observed\_count$) number of features to observe, and terminate the loop.
10: Update distribution $p(x_j)$ to a unit vector to reflect the observation made.
11: Update the decision δ_i for the task T_i that stream x_j belongs to, based on the new feature distribution $p(x_j)$ (Equation 4).
12: Update the Q_{Obs} values for the remaining unobserved streams belonging to task T_i (Equation 13).
13: $observed\_count \leftarrow observed\_count + 1$
14: **end while**
15: Update the Markov model for each stream based on observations made in this and previous time unit (add counts for newly observed transitions, and remove those expired out of the sliding window).

example, to compute $R^{obs}_{before}(c_k)$ in Eq 11 for a task with feature vector distribution $p(\vec{x})$, we need to know two sets of values.

- The risk value $R(c_i|\vec{x})$ for feature vector $\vec{x}$ can be obtained from the underlying Bayesian classifier, which computes an estimated posterior $P(c_i|x)$ from likelihood $P(x|c_i)$ and prior $P(c_i)$ and estimates risk accordingly[5].
- The movement distribution probability $p(\vec{x})$ for feature vector $\vec{x}$ can be obtained from the Markov models. Suppose $\vec{x}$ has k features, then the probability for the full feature vector is $p(\vec{x}) = \prod_{i=1}^{k} p(x_i)$, based on the assumption of feature movement independence. Here each $p(x_i)$ on an individual feature is computed using the corresponding Markov model. Suppose for feature x_i, the feature distribution at time $t-1$ is $p_{i_{t-1}}$, then

$$p_{i_t} = p_{i_{t-1}} K$$

where K is the state transition matrix for the Markov model of feature x_i.

We then compute the Expected Observational Risk by integrating over the domain of feature vector $\vec{x}$, which is discussed next.

Numeric Integration Over Feature Space To compute the Expected Observational Risk we need to integrate over the entire feature space of a task. This is computationally expensive if the task has a high dimension. To reduce the computational complexity we use *integration by sampling* as validated in [4] and in our own experiments.

In short, based on the independence in movement assumption, we perform 1-dimensional Monte Carlo sampling [15] on each feature based on its predicted data distribution, and then assembled the results from all features to form samples for the full feature vector, which can then be used to compute the expected risk as an un-weighted average. We will omit further details on this.

Markov Model Maintenance We separately maintain one Markov chain for each feature. If a feature has M distinct values, a matrix of $M \times M$ counters is maintained for the feature. Due to load shedding, we may not have consecutive observations on a particular feature to fill up the counters. We adopt an an-hoc method to force some consecutive observations in order to fill the counters, as used in [4], which will not be further discussed here. The dynamic nature of the streaming environment can also be addressed by building the Markov model on a sliding window of data, which we do not further discuss.

Feature Set Pruning to avoid Q_{obs} evaluation The most expensive step in BFF is to compute the metric Q_{obs} for each feature of each classification tasks, possibly repeatedly, as discussed below. To reduce the cost of the algorithm, we apply some heuristics to avoid evaluating the Q_{obs} of some features. Primarily, two heuristics are applied: 1) Avoid computing Q_{obs_j} metric (the gain of observational risk given the expectation of feature j) for features from tasks that have very low observational risk values to begin with. A threshold risk value is adaptively set, (e.g. the 20 percentile of the non-zero Q_{obs} values from the last window) and used to prune features from such low-risk tasks. 2) Further, we prune features whose Q_{obs} in the last window was below the threshold, and the overall risk value of the task has changed very little compared to last window. That is, even if a task has a overall risk that makes the threshold, we avoid features in the task whose observation is not likely to give rise to enough risk gains. Although the worst case is not affected, these heuristics effectively reduces the amortized average computational complexity in our experiments.

Algorithm Cost Analysis The most expensive step in BFF is to compute the metric Q_{obs} for each feature of each classification tasks. Without using the feature set pruning methods discussed above, suppose there are *n* tasks with *k* dimensions each (therefore there are a total of $N = n \times k$ streams), and out of them we have the capacity to observe *m* streams. Before we make any observation, we will perform a total of $O(N)$ computation of Q_{obs} metrics. Then after making each observation, we will only need to update metric values for $O(k)$ un-observed features for the affected task, which makes the total Q_{obs} update cost to be $O(m \times k)$. Therefore, each round we perform $[O(N) + O(m \times k)]$ computations of the Q_{obs} metric. With feature set pruning, the amortized average cost reduces a lot as confirmed by our experiments.

The sampling step of Q_{obs} computation, as discussed above for integration, only needs to be done once per time unit. Suppose we obtain h samples on each feature, the total cost of sampling is then $O(h \times N)$. h is usually a small number here, as 10-20 samples are usually enough to give a very good estimation in our experiments.

Maintaining the Markov models for each feature require $M \times M$ space complexity, and $M \times M$ time complexity for counter updates in each time unit. Therefore, we have a total of $N \times M \times M$ updates for Markov model maintenance.

4.3 A Lower-cost Hybrid

In this section we discuss an alternative approximation algorithm that further reduces the cost of metric computations complexity to the same level of task-based algorithms, while performing better than task-based algorithms.

Highest Variance of Worst Task (HVWT) Intuitively, instead of completely operate on features, this algorithm is a hybrid of task-based and feature-based algorithms, in which we pick a task first before picking a feature from the task. First, we pick the *worst* task that has the highest overall Observational Risk, by using Eq 11 which is computed on tasks. Then, instead of observing all the features in this *worst* task (as a task-based algorithm, such as LoadStar [4], will do), we only pick one *best* feature (in term of observation) from this task to observe. We then update the task's Observational Risk value after this observation, and start over again to pick the *worst* task and a *best* feature, and repeat this process until the capacity is reached.

To pick the *best* feature we utilize the following intuition. Frequently, a feature with a high variance in terms of movement destination will contribute more to the overall Observational Risk. For example in Figure 3, feature X_2 for task A has a high variance in movement, and observing which will result in a larger Observational Risk reduction than observing feature X_1. Intuitively, the higher the variance in movement, the more likely the destination will run across decision boundary, and therefore the larger its contribution to total Observational Risk. Of course a high variance does not always lead to a larger Observational Risk, e.g. in Figure 2 it is the lower-variance feature (X_1) in task C that contributes more to the Observational Risk, therefore we may not always be picking the *best* feature by this approximation.

Assuming feature movement patterns usually last for some period of time, the variance of movement for each feature can be computed once and reused in each time unit, only to be re-evaluated periodically. Therefore here

in each time unit we asymptotically avoid computing the $[O(N) + O(m \times k)]$ Q_{obs} metrics, and instead only do $O(n)$ computations of Expected Observational Risk for each task.

5 Experiment Evaluation

We apply feature-based load shedding on both synthetic and real-life data sets. Results indicate that the BFF algorithm out-performs both the random-shedding algorithm and the task-based shedding algorithm LoadStar[4] on multi-source classification tasks. In addition, the lower-cost hybrid HVWT algorithm appears to have a good trade-off between complexity and error performance.

5.1 Experiment Setups

We implement the load shedding and classification algorithms in Java. The experiments are carried out on a Linux machine with a P4 2.8GHz processor and 1GB of main memory. We use the Naïve Bayesian classifier, which has been shown to be highly effective in practice, as our base classifier, and we use a 0/1 loss function for risk computation. For ease of study we fix the number of input streams for each time unit, and compare the classification errors under different amount of load shedding. Because of the 0/1 loss function, classification error is simply computed as the percentage of mis-labeled data points. For the Monte Carlo sampling we use 10 sample points for each task.

5.2 Experiments on Synthetic Datasets

We generate data for 25 classification tasks each with K features (i.e. K different streaming inputs), thus for a total of $25 * K$ input streams. For ease of study, the tasks share the same two-class classification model. Due to the *naïve* assumption, the class models on each feature are assigned independently. Half of the K features for each task are assigned with the following class model: $p(x|+) \sim N(0.3, 0.2^2)$, $p(x|-) \sim N(0.7, 0.2^2)$, where $N(\mu, \sigma^2)$ is Normal Distribution with mean μ and variance σ^2. The other half features in each task are assigned with the following class model: $p(x|+) \sim N(0.7, 0.2^2)$, $p(x|-) \sim N(0.3, 0.2^2)$. Then the *real* class of each generated point is assigned using the class with the higher joint posterior probability.

For data point movements, we use a random walk model:

$$x_t = x_{t-1} + \epsilon, \text{ where } \epsilon \sim N(0, \sigma^2)$$

To have a mixture of different movement variances, half of the K features in each task are assigned with a σ value of 0.3, and the other half are assigned with a σ value of 0.005. Therefore the features in the same task could have very different movement variances.

Data are generated for 10000 time units on all the streams, the first 5000 time units are used for training the classifier and the Markov models, and the rest are used for testing. We omit sliding window management for Markov models in our experiments, since its effectiveness in adapting to changing movement patterns under this type of load shedding settings has already been validated elsewhere [4].

Quality of Classification Figure 5 shows the quality of classification under different load shedding percentages for load-shedding algorithms with different quality metrics(K=4, i.e. 4 features per task). Since random shedding does not use any intelligence in selecting features for observation, we use random shedding as the *baseline* for comparison. The horizontal axis shows the percentage of load that is shed from observation, and the vertical axis shows the relative error compared to the error of random-shedding (i.e. $\frac{Error_{algorithm}}{Error_{random}}$). We see that the feature-based greedy algorithm utilizing metric Q_{Bayes} (line C) performs better than the task-based load shedding method LoadStar (line B), while the BFF algorithm (line D), which is feature-based and specifically targeting the Observational Risk, outperforms all other methods. The BFF algorithm achieves more than 45% improvements over random load shedding when the amount of shedding is about 40% to 50%. When the amount of shedding further increases, the improvement drops as prediction becomes less accurate.

Figure 6 validates the hybrid algorithm HVWT (line E)[7]. This approximation algorithm outperforms LoadStar

[7] We try to keep the labeling of algorithms consistent across different figures, therefore here we have kept the labels B and D, and added label E.

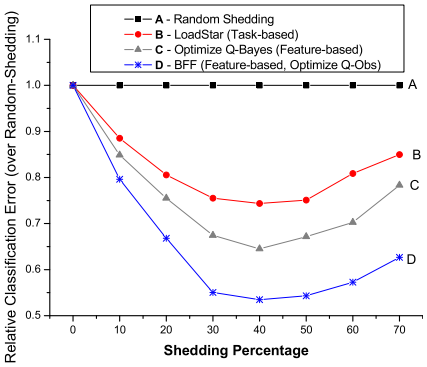


Figure 5: Comparison of Task-based and Feature-based Methods on Synthetic Data

while achieves classification error close to that of BFF.

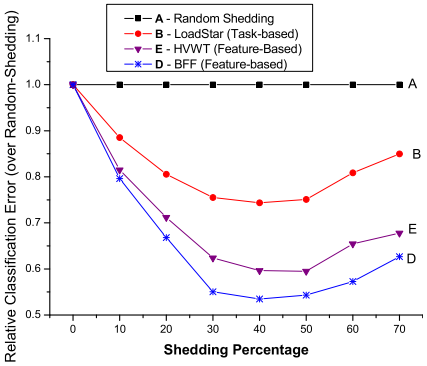
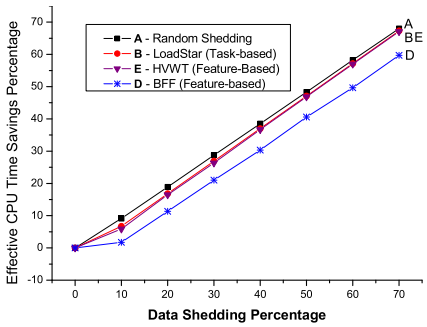


Figure 6: The Approximation Algorithm HVWT on Synthetic Data

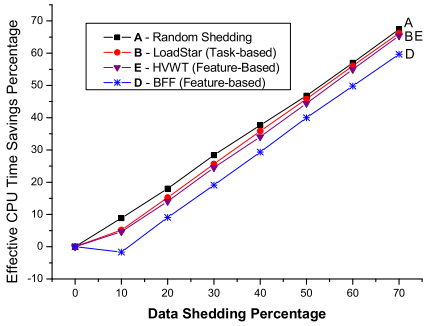
CPU Cost To study the cost of different algorithms, we measure the *effective CPU time saving* under different load shedding conditions. Because of computation overhead, when we shed x% of data from observation, we actually achieve a total CPU cost saving that is less than x%. Therefore, we measure the total CPU time required under load shedding, and divide it by the total CPU time required without load-shedding. This ratio is then the *effective CPU time saving* achieved by load shedding. Therefore, under a given load-shedding amount, the higher this ratio is, the less overhead is commanded by the algorithm.

As discussed in Section 1, our algorithm applies to the case when data observation is associated with high cost (e.g. audio/video streams that have high feature extraction cost, or sensor data that has a high communication cost). Therefore, to simulate a realistic situation and study the costs of the algorithms, in the experiments we assign an observation cost c to each data observation (i.e. It costs c as in CPU time to observe a data source). In particular, in the experiments in Figures 7, c is set to 5 milliseconds per data source observation. This observation cost is reasonable in many situations. For example, suppose our algorithm detects alert situations by first tracking the movement of objects in video frames, and then classifying whether danger is present, depending on the object position in the frame. The object recognition step for the video frames thus becomes a pre-processing step for the danger classification. Such recognition process easily takes tens of milli-seconds in practice. As an example, suppose the object we are tracking is a human face, in [17] it is shown that using state-of-the-art technology, it takes 67 milli-seconds to recognize face on a 384x288 image. (In comparison, in our experiments, it takes about 1 millisecond to randomly select a feature, and then predict its value and classify the corresponding task. Therefore, the observation cost can be a much more significant cost compared to classification.)

Figure 7(a) ($K = 2$, i.e., two-features per task) and 7(b) ($K = 4$, i.e., four-features per task) shows the effective CPU time savings under different load shedding conditions. The result shows that LoadStar and HVWT introduce an overhead that is quite close to each other, with both costs a little higher than random shedding. Nevertheless, both of these two algorithms result in quite small overheads. For example, if the data tuple shedding is 10%, the effective CPU time saving of these algorithms is in the range of (5-7)%. On the other hand, the BFF algorithm shows a higher overhead, where CPU savings from the first 10% tuple shedding is basically consumed by the algorithm overhead, i.e., a 20% tuple shedding roughly achieves a 10% CPU time saving for the BFF algorithm.



(a) Two-features per Task



(b) Four-features per Task

Figure 7: The CPU Cost of Algorithms

5.3 Real Life Experiments: Traffic Jam Prediction

For real-life data, we use datasets exported from National Center for Data Mining database of Illinois IDOT Highway Sensors [1]. Aggregated traffic information of average speed, volume etc, are collected by sensors located along highways (readings about 5 minutes apart). We devise the following streaming application: based on the hour of day, the traffic condition on a highway segment, as well as the traffic conditions on the adjacent segments, predict whether there will be a *long traffic jam* on this segment. We simplistically define a long traffic jam as "average speed $<$ 15mph for a consecutive 30 minutes or longer with no gaps". Real traffic data are labeled either as *in long traffic jam* or *not in long traffic jam* according to this criteria, and used to train a separate Naive Bayes classifier for each highway segment. The idea is that, the traffic conditions (e.g. average speed) on highways should follow some stochastic process that can be reasonably predicted by a Markov model, and therefore we can perform load shedding based on these predictions.

For the experiments, we select 4 highway segments (4 tasks), each task with a total of 64 features, including i) the traffic information at current time unit for this segment and the 2 segments before and 2 segments after this segment. ii) the traffic condition on the these same 5 segments during the last 3 time units.

Out of the 64 features per task, only the 16 for the current time unit needs to be modeled and predicted. The other 48 features are historical data (for the last 3 time units) that are simply copied over from the observations/predictions at the previous time units. Therefore, under this setting we have 16 streams per task for load shedding, with a total of 64 data streams overall for 4 tasks. We used 3-weeks worth of aggregated traffic data (collected roughly once every 5 minutes) to train Naive Bayes classifiers, and use one-week worth of data for testing. The *long traffic jam* prediction is rather successful under this setting, with a base error rate of about 4%.

Figure 8 shows the experiment comparing 4 load shedding algorithms for the traffic prediction task. Very similar to synthetic data, we see that the BFF algorithm with metric Q_{Obs} (line D) gives the best performance, while the approximation algorithm HVWT outperforms LoadStar and approaches the performance of the BFF algorithm.

6 Related Works

The task-based load shedding algorithm LoadStar [4] studies a special case of the stream classification problem, where every task only has one input data stream. In our paper, we focus on the more general case where each task may have multiple input sources, and devise feature-based metric and algorithms accordingly.

Load shedding mechanisms for data stream queries has been studied for Data Stream Management Systems (DSMS). These systems generally either employ a random-dropping mechanism [2, 9, 16], rely on user-

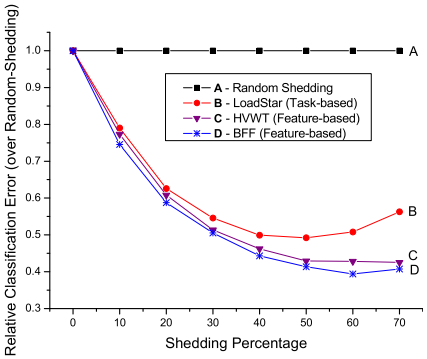


Figure 8: Comparison of Task-based and Feature-based Methods on Traffic Prediction

provided static QoS metric [9], or build an adaptive feedback loop for tuple latency based on control theory [10]. These methods do not address the quality requirements of classification tasks, where the quality measures are non-static and task-dependent.

Adapting classifiers for streaming data is another related area [18, 12, 13, 14], which usually studies one-pass incremental algorithms, builds data synopsis, or adapts classifiers to concept-shifts. Our work instead focuses on intelligently dropping, not approximating, input data under overloaded conditions.

Another related area is distributed data streams querying [7, 6], which focuses on cost savings across a distributed network. Here we focus on a multi-source task setting, where only the server has the full knowledge to decide which data to drop.

7 Conclusions

In this paper we adopt a Bayes Risk based approach toward the multi-source classification problem. We performed a full analysis of Bayes Risk and propose a risk measure for load shedding - the Observational Risk. We devise the BFF algorithm for feature-based load shedding and its lower-cost approximation algorithm HVWT, and use both synthetic and real-life data to validate the performance of the algorithms.

References

[1] The Pantheon Gateway Testbed project. http://highway.lac.uic.edu.

[2] B. Babcock, M. Datar, and R. Motwani. Load shedding techniques for data stream systems, 2003.

[3] Brian Babcock, Mayur Datar, and Rajeev Motwani. Load shedding for aggregation queries over data streams. In *ICDE*, 2004.

[4] Yun Chi, Philip S. Yu, Haixun Wang, and Richard Muntz. Loadstar: A load shedding scheme for classifying data streams. In *SIAM DM*, 2005.

[5] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000.

[6] Ankur Jain et. al. Adaptive stream resource management using kalman filters. In *SIGMOD*, 2004.

[7] C. Olston et. al. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD*, 2003.

[8] Mohamed M. Gaber et. al. Mining data streams: a review. *SIGMOD Rec.*, 34(2), 2005.

[9] N. Tatbul et. al. Load shedding in a data stream manager. In *VLDB*, 2003.

[10] Yi-Cheng Tu et. al. Control-based quality adaptation in data stream management systems. In *DEXA*, 2005.

[11] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *ACM SIGMOD Record*, 32(2):5–14, 2003.

[12] Sudipto Guha and Nick Koudas. Approximating a data stream for querying and estimation: Algorithms and performance evaluation. *ICDE*, 2002.

[13] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *KDD*, 2001.

[14] Rouming Jin and Gagan Agrawal. Efficient decision tree construction on streaming data. In *KDD*, 2003.

[15] J.S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer, 2001.

[16] R. Motwani, J. Widom, A. Arasu, B. Babcock, M. Datar S. Babu, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, approximation, and resource management in a data stream management system.

[17] Paul Viola and Michael J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, 2004.

[18] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *SIGKDD*, 2003.